VU Research Portal

Safe and private data sharing with turtle: friends team-up and beat the system

Popescu, B.C.; Crispo, B.; Tanenbaum, A.S.

published in
Proc. 12th Cambridge International Workshop on Security Protocols April 2004.
2004

document version
Publisher's PDF, also known as Version of record

Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System

Bogdan C. Popescu
Vrije Universiteit
Amsterdam, The Netherlands
bpopescu@cs.vu.nl

Bruno Crispo
Vrije Universiteit
Amsterdam, The Netherlands
crispo@cs.vu.nl

Andrew S. Tanenbaum
Vrije Universiteit
Amsterdam, The Netherlands
ast@cs.vu.nl

Abstract

In this paper we describe Turtle, a peer-to-peer architecture for safe sharing of sensitive data. The truly revolutionary aspect of Turtle rests in its novel way of dealing with trust issues: while existing peer-to-peer architectures with similar aims attempt to build trust relationships on top of the basic, trust-agnostic, peer-to-peer overlay, Turtle takes the opposite approach, and builds its overlay on top of pre-existent trust relationships among its users. This allows both data sender and receiver anonymity, while also protecting **each and every** *intermediate relay in the data query path. Furthermore, its unique trust model allows Turtle to withstand most of the denial of service attacks that plague other peer-to-peer data sharing networks.*

1 Introduction

Freedom to exchange information derives from the freedom of speech; unfortunately, there are many countries where this basic human right is not guaranteed. Turtle is a peer-to-peer data sharing architecture that makes very hard to restrict the freedom to exchange information by either technical or legal means.

When designing Turtle, we were inspired by the way people living under oppressive regimes [1] share information deemed "hostile" by their government (this can be books, newsletters, video and audio recordings, or even political jokes). Because of the potentially very serious consequences raising from being caught possessing/distributing such material, no single individual is willing to share it, except with close friends. Experience has repeatedly shown that, even in the most repressive environments, this "friends-to-friends" delivery network is remarkably effective in disseminating information, with relatively little risks for the participating parties; if one chooses his friends carefully, the chance of being caught doing the forbidden exchanges becomes very small.

The idea behind the Turtle is to take this "friend-to-friend" exchange to the digital world, and come up with a peer-to-peer architecture allowing private and secure sharing of sensitive information between a large number of users, over an untrusted network, in the absence of a central trust infrastructure.

The rest of this paper is organized as follows: in Section 2, we give a high level description of the Turtle architecture, including the protocols for query propagation and result retrieval. In Section 3 we look at technical, security and ethical implications of our design choices. In Section 4, we review related work, and in Section 5 we present our conclusions.

2 The Turtle Architecture

To bring the discussion to a more formal level, we will introduce a system model. For the Turtle architecture, this consists of a large set of nodes $\mathcal{N}$, and a large set of data items $\mathcal{D}$. We assume that behind each Turtle node i there is a human user (the node's **owner**) who has a subset D_i of all items in $\mathcal{D}$, and is interested in obtaining more. However, a user owning a node i is willing to share his data items only with nodes owned by people he trusts - we denote this as $i's$ **friends subset** - F_i. We assume the friendship relation is commutative, for any two nodes i and j, if i is in F_j, then j is in F_i. However, friendship is not transitive (the friend of a friend is not

[1] One of the authors was born in a country where the government's track record on civil liberties used to be less than stellar.

automatically a friend).

Each data item d has an **attribute set** A_d associated with it. The attribute set consists of a number of *attribute=value* pairs describing certain properties of the data item, and are used when evaluating user queries. These are logical expressions consisting of a number of *attribute=value* pairs, connected using the *AND, OR* and *NOT* logical operators. A data item matches the query if the *attribute=value* pairs in its attribute set satisfy the logical condition in the query expression.

Each user establishes a cryptographically secure connection between its Turtle node and all the nodes in its friends subset. Since there is no central trust infrastructure, the shared secrets needed to establish these secure connections have to be agreed-upon by out-of-band means (this can be done using common knowledge based on common past experiences - after all owners of friend Turtle nodes are assumed to be friends in the real-life!). Once established, the inter-friends secure communication links are used for exchanging data items and propagating user queries.

Users search for new data items by sending queries to the Turtle network. The user starts by introducing a query expression and a query depth through the query interface of its Turtle node. The node then creates a 128 bit **query ID**, by generating a 64 bit random number and appending it to the most significant 64 bits of the SHA-1 hash of the query expression (treated as an ASCII string). In this way, the probability that two distinct queries will have the same query ID is extremely small. Once it has the query ID, the node constructs a query packet containing the query expression, the query ID, and a hop count, initially set to the query depth. The query packet is then broadcast over the "friendship" links up to the desired query depth. Upon receiving a query packet, a Turtle node first evaluates the query expression against the attribute sets of all the data items in its data subset. If matches are found, the node reports them back to node that has forwarded (possibly originated) the query. Furthermore, the node decrements the hop count in the query packet, and if the count is still positive, the packet is further forwarded to all the node's friends (except the one from which the packet came).

We can see that propagating a query in the Turtle network generates a *query broadcast tree* rooted in the node originating the query, tree that follows the trust relationships among the Turtle users. The query broadcast tree is also used for delivering query answers, which travel hop by hop up the tree until they reach the root. In order to match queries with answers, each node maintains a query table with queries it has forwarded but for which the query answer process has not yet completed. Each query table entry corresponds to a query broadcast tree the node is part of; table entries are indexed by query ID, and store the address of the node's parent in the tree, the time the query has been received, and a result section, storing all the response packets the node has received from its children nodes in the tree.

A query response packet consists of the address of the responder, the *Final* bit, the query ID, a response hop count, and possibly a *response payload* consisting of number of data attribute sets. The *Final* bit is used for differentiating between partial and final answers. A node receiving a positive answer (query hit) from one of its children in the query broadcast tree, will immediately report it to the parent. A node indicates it has no more answers to forward by sending his parent a response packet with the *Final* bit set. This can happen in the following circumstances:

- The node receives a query packet with a query ID that matches one already present in the query table (we call this a *collision*). Since it is very unlikely two different nodes will generate the same query ID, the most likely cause for the collision is a cycle in the friendship graph which has routed the same packet back.
- A node receives a query packet with a zero hop count (no more forwarding necessary). The node responds with a final packet which also includes the attribute sets of the local data items matching the query (if there are any).
- The node has received final answers from all its children in the tree, and has finished processing them.

A response packet with the *Final* bit set to zero is a partial answer. The following rules apply to partial answers:

- A node receiving a query matching some of the elements in its data set creates a partial answer with the response hop set to zero, and a payload consisting of the attribute sets of all data items that match the query.
- A node receiving a partial answer from one of its children in the tree **changes the responder's address in the packet to its own address**, increments

the response hop count, and forwards it to its parent node. The node also keeps a copy of the original response packet (as received from its child) in the response section of the query table (this is needed for the data retrieval phase, as we will see shortly).

The query completes after the originating node has received final answers from all its friends. At this point, the originating node has also accumulated all partial answer packets. The node then sorts through all these partial answers to identify all distinct data attribute sets; these are then presented to the user, much in the same way as the results of a Web search engine query (they can be ranked based on frequency, or hop distance). Once the user selects the result she is interested in, the node can start the *data retrieval* phase.

The retrieval phase consists of selecting a *retrieval path* and propagating the query result along that path. For a given data element d (identified by its attribute set A_d), the retrieval path is the shortest path in the query tree between the root and a node that has d. This path is determined hop by hop, starting with the root node which searches through all response packets and selects the one that has A_d in the payload, and the smallest response hop count. The root then asks the friend from which the selected response packet has been received to retrieve the A_d data item. The friend follows a similar procedure to find the next hop in the retrieval path, and so on until the retrieval request reaches the node that has the actual data item. The data item is then sent to the requester, following (hop by hop) the the retrieval path in reverse order [2].

3 Discussion

There are two basic assumptions we make when proposing the Turtle architecture. First, we assume that continuous, high-speed Internet connections will become ubiquitous in the near future. Looking at current trends, which show increasing DSL/cable modem penetration in the consumer market, not to mention the ever-increasing wireless "umbrellas" that cover large parts of big cities, this first assumption seems very reasonable. The second assumption is that for sufficiently large social communities (a college campus, a country, the world), "friendship" relationships form fully connected graphs. Validating this assumption would obviously involve large-scale sociological experiments, which are beyond the scope of this paper. However, based on the famous "six degree of separation" experiment [15], the moderate success of the PGP infrastructure, and, more recently, the explosive popularity of the "Friendster" service [1], we have reasons to believe that for relationships involving moderate amount of trust, it is very likely to achieve full "friendship" graph connectivity.

[2] this is slow but safe, hence the name "Turtle"

3.1 Technical Considerations

We expect that our main application scenario (dissidents exchanging sensitive information) would mostly involve small data items. For this reason, we have designed Turtle as a packet routing overlay. However it should be straightforward to re-design it as a circuit switching network, so it could accommodate large data transfers. The only significant change in the protocol would concern the retrieval path, where nodes would have to establish a "virtual tunnel" connecting the source and the destination. This would also require dealing with issues such as flow control, buffering and quality of service.

Second, overall system performance can be greatly improved if data items are cached. This can be complemented by "gossip-like" mechanisms that allow nodes to quickly disseminate information about which data items are popular, in order to implement smarter cache expulsion algorithms.

Finally, because the unique trust properties of the overlay (only nodes that trust each other directly interact), it is very easy to enhance Turtle with an economic model that would encourage cooperation and sharing. For example, when sending back a response packet, a node can also include a price tag for supplying the given item, with each node in the broadcast tree adding his "relay fee" to the price tags received from its children. The query initiator can then use the final price tag as an additional selection criteria when deciding which data item to request. These payments can be aggregated over longer intervals, by having Turtle nodes keeping track of the amount owed to/owed by friend nodes. A node can then periodically report the "balance sheet" to its owner, who can settle the matter with his friends by out of band means (e.g. cash exchange).

3.2 Security Implications

From a security point of view, the Turtle architecture raises a number of interesting points:

First of all, Turtle offers good query initiatior and query result anonymity: both initiator and responder are known only by their respective friends subsets (trusted nodes). With small modifications in the query/result routing protocol - namely removing the hop counts - it is also possible to achieve complete sender/receiver anonymity. Because all information exchange is done over encrypted channels, the only way for an adversary to link a query initiator to a responder is through traffic analysis. However, there are well known techniques for protecting against traffic analysis [12, 7], which can be easily incorporated in our basic query/result routing protocol.

Second, the Turtle network is immune to the "Sybil" attack [6]. Even if a powerful adversary is able to create a large number of malicious Turtle nodes, the effect these nodes have on the correct functioning of the system is minimal, unless the attacker is also able to infiltrate his nodes in the friends sets of correct nodes (but this would require a lot of social engineering!).

Third, Turtle exhibits a very desirable fail-mode property - "confined damage" - meaning that a security break in one correct Turtle node only affects a small subset of all correct nodes in the system (in this case the node itself plus its friends subset).

Finally, due to the way the Turtle overlay is organized, denial of service attacks typical for a peer-to-peer network - such as malicious routing [2], content masquerading (content that does not match its description), bogus query hits (a node answering positive to a query even when it does not have any matching content), and aborted transfers - are much less likely to happen. Because all direct interactions take place between nodes controlled by people who trust and respect each other ("friends"), we expect incentives for random malicious behavior to be very much reduced.

3.3 Ethical Implications

Turtle allows private sharing of data, and protects data providers, data consumers as well as intermediaries from the potential negative consequences stemming from being identified as participants in this data exchange. The main motivation for our work is providing citizens living under oppressive regimes with a safe, private information sharing network. However, it is clear that this same technology can be used for activities of dubious ethical value - such as illegal sharing of copyrighted digital products.

There is a lot of debate regarding the legality/morality of sharing copyrighted information over peer-to-peer networks. It has already been pointed out [5] that the benefits copyright laws have brought to society in the printing press age no longer apply in the case of digital content. It remains to be seen how copyright laws will be changed to adapt to digital technology; nevertheless, it is our opinion that widespread use of fast and efficient file sharing networks such as Kazaa and Gnutella (where almost 100% of the traffic deals with illegal sharing of copyrighted material), if left unchecked, will eventually drastically reduce the incentives for artistic creativity. However, using a Turtle-like network for such activities, will have a less drastic impact. Because of the way Turtle is designed, the speed of the data retrieval is dictated by the bandwidth capacity of the slowest link in the retrieval path. Thus, using Turtle to share large items will be most likely considerably slower than in Kazaa and Gnutella. This "un-ease of use", combined with the fact that at any moment data is only exchanged between friends (thus, the only way to legally pursue copyright infringers would involve morally despicable strategies, such as forcing people to "rat" on their friends), may actually help in defining a new legal framework on what constitutes "acceptable sharing" and "fair use."

4 Related Work

What makes Turtle unique is the bottom-up way it builds its overlay starting from *pre-existing* trust relationships among users. To the best of our knowledge, there are no other peer-to-peer systems that employ this technique.

However, the idea of indirect routing of requests and results in order to achieve sender/receiver anonymity has been around for some time. Chaum [3] has first suggested it for building a mix network for anonymous e-mail delivery. Crowds [12] builds a peer-to-peer mix network for anonymous Web browsing. The same idea is more recently employed in [11] and [7] for building general-purpose anonymizing network layers.

In the context of peer-to-peer data sharing networks, indirect routing for queries and results is being employed by Freenet [4] with the aim of creating an "uncensorable and secure global information storage system". However, a Freenet node acting as a relay in a "forbidden exchange" can only achieve a limited degree of plausible deniability. This relay node exposure, combined with the lack of a-priori trust relationships among

interacting nodes, makes Freenet a hard sell for a "dissidents network." The same considerations apply to other anonymizing, censorship-resistant peer-to-peer systems [13, 8, 10, 14, 9]: none of them manages to offer an acceptable level of protection to **each and every** entity in the request/retrieval path.

5 Conclusion

In this paper we have described Turtle, a peer-to-peer architecture for safe sharing of sensitive information. In order to achieve strong privacy guarantees, Turtle organizes the data sharing overlay on top of pre-existing user trust relationships. This protects the privacy of both data senders and receivers, as well as the intermediate relay nodes that facilitate the data exchange. Furthermore, Turtle is resistant to most of the denial of service attacks that plague existing peer-to-peer data sharing networks.

As for directions for future work, we have already mentioned possible extensions of the basic query protocol to support retrieval of large data items (through hop-by-hop virtual circuits) and economic models that would encourage cooperation and sharing. It would also be interesting to look at ways to associate sensitivity levels to data items and different trust levels to different friends, which would allow node owners to specify more complex security and privacy policies regarding the data items they share.

References

[1] Friendster Web Site. http://www.friendster.com.

[2] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proc. OSDI'02*, Dec. 2002.

[3] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Comm. of the ACM*, 24(2), 1981.

[4] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. Int. Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, pages 46–66, 2001.

[5] James A. Dewar. The Information Age and the Printing Press: Looking Backward to See Ahead. http://www.rand.org/publications/P/P8014/P8014.pdf.

[6] J. Douceur. The Sybil Attack. In *Proc. of the IPTPS '02 Workshop*, Mar. 2002.

[7] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proc. of the 9th ACM Conf. on Computer and Communications Security*, November 2002.

[8] S. Hazel and B. Wiley. Achord: A Variant of the Chord Lookup Service for Use in Censorship Resistant Peer-to-Peer Publishing Systems. In *Proc. of the IPTPS '02 Workshop*, Mar. 2002.

[9] A.D. Rubin M. Waldman and L.F. Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system . In *Proc. 9th USENIX Security Symposium*, pages 59–72, August 2000.

[10] R. Dingledine, M.J. Freedman, D. Molnar. The Free Haven Project: Distributed anonymous storage service. In *Proc. Int. Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, pages 67–95, 2001.

[11] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE J. on Selected Areas in Communications*, 16(4), 1998.

[12] Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[13] A. Serjantov. Anonymizing Censorship Resistant Systems. In *Proc. of the IPTPS '02 Workshop*, Mar. 2002.

[14] M. Waldman and D. Mazieres. Tangler: a censorship-resistant publishing system based on document entanglements. In *Proc 8th ACM Conf. on Computer and Communications Security*, pages 126–135, 2001.

[15] Duncan J. Watts. *Small Worlds, The Dynamics of Networks between Order and Randomness*. Princeton University Press, Princeton, NJ, 1999.